System Synthesis from AADL using Polychrony

Yue Ma, Huafeng Yu, Thierry Gautier, Jean-Pierre Talpin, Loïc Besnard, Paul Le Guernic

System Synthesis from AADL using Polychrony

Yue Ma Huafeng Yu Thierry Gautier Jean-Pierre Talpin Loïc Besnard Paul Le Guernic
INRIA Rennes/IRISA/CNRS, Campus de Beaulieu, 35042 Rennes Cedex, France
Email: {firstname}.{lastname}@inria.fr Loic.Besnard@irisa.fr

***Abstract*—The increasing system complexity and time to market constraints are great challenges in current electronic system design. Raising the level of abstraction in the design and performing fast yet efficient high-level analysis, validation and synthesis has been widely advocated and considered as a promising solution. Motivated by the same approach, our work on system-level synthesis is presented in this paper: use the high-level modeling, domain-specific, language AADL for system-level co-design; use the formal framework Polychrony, based on the synchronous language SIGNAL, for analysis, validation and synthesis. According to SIGNAL's polychronous model of computation, we propose a model for AADL, which takes both software, hardware and allocation into account. This model enables an early phase timing analysis and synthesis via tools associated with Polychrony.**

***Index Terms*—AADL; Polychrony; Synthesis;**

I. INTRODUCTION

In the application domains such as avionics and automotive, electronic systems are generally an integral part that needs to satisfy safety-critical requirements. As a result of fast development of hardware and software in recent years, the increasing system complexity becomes a great challenge. For example, more and more software/hardware subsystems or components are integrated together to provide general-purpose chips due to flexibility and size requirements. However, the resulting complexity may lead to system reliability issues. Moreover, system validation is engaged in a large proportion of time in the development cycle, which places a limit on time to market.

As a promising solution to the previous issues, raising the level of abstraction in the design and performing fast yet efficient high-level validation and synthesis attract the attention of both industry and academia. These approaches efficiently help to reduce the design time cycle and complexity. For instance, high-level modeling and analysis languages are used for fast virtual prototyping and design space exploration before a real physical implementation. We are interested in languages that can describe both software and hardware, and languages that are based on rigorous semantics so that efficient formal validation can be performed.

Architecture Analysis and Design Language (AADL [21]) is a SAE (Society of Automotive Engineers) standard. AADL enables to identify the structural components, and eventually express properties of the whole architecture. At the AADL specification level, an abstract of software application is distributed onto a set of execution platforms without necessarily having a physical implementation of the system at hands.

Synchronous languages can significantly ease the modeling, programming and validation of embedded systems [7]. SIGNAL is a domain-specific, synchronous data-flow language, dedicated to embedded and real-time system design [15]. While being declarative like SCADE or Lustre [12], and not imperative like Esterel [10], its multi-clocked model of computation (MoC) stands out by providing the capability to design systems where components own partially related activation clocks. This polychronous MoC, called Polychrony [16], also provides the mathematical foundation to define a notion of behavioral refinement. Behavioral refinement is the ability to model a system from the early stages of its requirement specifications (properties) to the late stages of its synthesis and deployment (interconnected functions) by iterative upgrade with correctness-preserving, automated or manual, program transformations.

In order to support system level virtual prototyping, early-phase analysis and validation of component-based embedded systems, we define a model of the AADL based on the polychronous MoC of the SIGNAL programming language [16]. Synthesis is performed on this model to obtain a simulation model. On one hand, the resulting model has a formal specification allowing formal verification. On the other hand, real-time characteristics can be associated with this model so that profiling can be carried out. The main difficulty is to model AADL temporal properties into a polychronous model. For instance, time-related concepts in AADL, such as synchronization, delay, period, etc., are abstracted by discrete logical clocks, which can be independent or dependent. Thus, AADL time domain is mapped onto SIGNAL clocks. Independent or dependent clocks, such as clocks of different threads and clocks of processors, are partially correlated by schedulers and allocation specification for simulation afterwards.

Outline. Section II presents some background on AADL and SIGNAL. Section III introduces our approach of system synthesis from AADL specifications, by describing an interpretation of AADL into Polychrony. Some related works are addressed in Section IV, and conclusion is drawn in Section V.

II. BACKGROUND: AADL AND POLYCHRONY

A. AADL

The purpose of the SAE AADL standard is to provide a standard and sufficiently precise way of modeling the architecture of an embedded real-time system, to permit analysis of its properties, and to support the predictable integration of its implementation. As an architecture description language,

AADL describes the structure of such systems as an assembly of software components executed on execution platforms.

AADL adopts component-based paradigm for system description. To model complex embedded systems, AADL provides three distinct sets of component categories: *application software*, *execution platform* and *composite* components. *Data, subprograms, threads*, and *processes* collectively represent application software, they are called *software* components. *Execution platform* components support the execution of threads, the storage of data and code, and the communication platforms. *Systems* are called *composite* components. They permit *software* and *execution platform* components to be organized into hierarchical structures with well-defined interfaces.

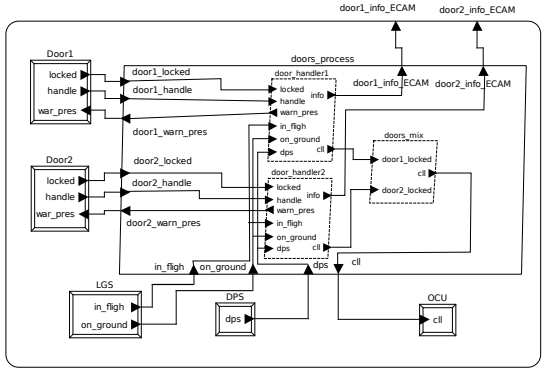

Fig. 1. AADL graphical example.

We show in Figure 1 part of the AADL specification of a SDSCS system (Simplified Doors and Slides Control System) [23]. A *doors_process* interacts with two devices (*Door1* and *Door2*). Three threads inside this process perform the computation for the door management. The AADL thread component and its connections will be presented in more detail in Section III.

Type and implementation: AADL components are defined through *type* and *implementation* declarations. A component *type* represents the functional interface of the component and externally observable attributes. The *implementation* describes the contents of the component, specifies an internal structure in terms of *subcomponents*, as well as *connections* between the features of those subcomponents. AADL allows a *type* associated with zero, one or more *implementations*.

Connections: A *connection* is a linkage between component features, that represents the communication of data and control between components.

Properties: A *property* provides information of the component that apply to all instances of this component, unless overwritten in implementations or extensions.

B. SIGNAL *language and Polychrony*

SIGNAL, based on the polychronous model of computation [16], is a data-flow language that allows the specification of multi-clocked systems. SIGNAL handles unbounded series of typed values $(x_t)_{t\in\mathbb{N}}$, called *signals*, denoted as x. Each signal is implicitly indexed by a logical clock indicating the set of instants when the signal is present, noted $\hat{x}$. At a given instant, a signal may be present where it holds a value, or absent (denoted by $\perp$).

$P, Q \ ::= \ x := y\ f\ z \mid P|Q \mid P/x$

In SIGNAL, a process (written P or Q) consists of the synchronous composition (noted $P|Q$) of equations (written $x := y\ f\ z$) over signals x, y, z. The process P/x restricts the lexical scope of the signal x to the process P. An equation $x := y\ f\ z$ defines the output signal x by the result of the application of operator f to its input signals y and z. Several basic operators are listed in the following:

- **Delay "$"** gives access to the value of a signal at its previous time sample. $y := x\$1\ init\ c \stackrel{def}{\equiv} (\forall t\ x_t = y_t = \perp) \vee (\exists m = min\{t \geq 0 \mid x_t \neq \perp\}, y_m = c \wedge \forall t\ (x_t \neq \perp \Leftrightarrow y_t \neq \perp) \wedge (t > m \Rightarrow (y_t = x_{pred(t)}$ where $pred(t) = max\{k < t \mid x_k \neq \perp\})))$.
- **Sampling "when"**: the equation $y := x\ when\ b$ defines y by x when b is present with the value $true$. $y := x\ when\ b \stackrel{def}{\equiv} y_t = x_t$ if $b_t = true$, else $y_t = \perp$.
- **Merging "default"** : the SIGNAL expression $z := x\ default\ y$ merges the signals x and y with a priority to x at any logical instant where at least one is defined. $z := x\ default\ y \stackrel{def}{\equiv} z_t = x_t$ if $x_t \neq \perp$, else $z_t = y_t$.
- **Cell "cell"**: the cell operation $y := x\ cell\ b$ repeats signal x on the instants of boolean signal b. The result y contains all values of x, and the value of previous x when b is $true$. $y := x\ cell\ b \stackrel{def}{\equiv} y_t = x_t$ if $x_t \neq \perp$, else $y_t = y_{pred(t)}$ if $b_t = true$, where $pred(t) = max\{k < t \mid y_k \neq \perp\}$, else $y_t = \perp$.

SIGNAL is associated with a design environment, Polychrony [13], which provides a formal framework for the system modeling at a high level of abstraction, design validation, as well as simulation for deterministic specifications.

III. CONTRIBUTION: SYNTHESIS FROM AADL MODELS

The main difficulties, when using Polychrony for the modeling and synthesis of embedded systems specified in AADL, are due to their different timing semantics. SIGNAL has a notion of logical time, the progression which must be made explicit in computation (with a *delay* operator over flows), as well as in communications (thanks to bounded FIFOs). Thus a SIGNAL application is ultimately composed of atomic "instantaneous" actions, logically timed as clocks over a partial ordered domain; actual duration of an atomic action is abstracted as a null duration. Conversely, AADL does not provide implicit instantaneous atomic actions: execution of a thread takes place between *dispatch* and *complete* events defining logical time intervals during which several occurrences of flows can be read and written. This logical time is related to physical time units in an application description. We propose solutions in our modeling so that AADL time domain is mapped onto SIGNAL clocks. The guiding principle is to represent a non-instantaneous AADL action as an instantaneous SIGNAL action nested in a container that provides "waiting" stations. This is done with no change in the semantics of the AADL

description. The physical time progressions are represented as (non-deterministic) SIGNAL processes. A high-level view of system synthesis is given in this section. Three stages are presented in the process, which include: modeling AADL software components in Polychrony, addition of scheduler models from processors and partition models from component allocation, and system integration.

A. From abstract logical time to concrete simulation time

AADL takes into account computing latencies and communication delays, allowing to produce data of the same logical instants at different implementation instants. Those instants are precisely defined in the port and thread properties. To tackle this problem, we keep the ideal view of instantaneous computations and communications, moving computing latencies and communication delays to specific "memory" processes, that introduce delays and well suited synchronizations.

1) Modeling computation latencies: A main feature of synchronous programs is the logical instantaneous execution, with respect to logical time: the outputs are immediately generated when the inputs are received. While in AADL, a thread may perform a function for a specified time interval. The output is available and transferred to other components at a time specified by *Output_Time* timing property. Therefore, modeling AADL in the polychronous framework requires some adapter for interfacing abstract logical time and concrete simulation time.

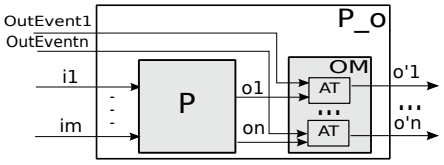

Fig. 2. Modeling a time consuming task.

With a process P, we associate a process P_o, the output o' of which is the value of the output o of P delayed until its output time occurs, represented by the input event signal *OutEvent* (Figure 2). An additional output memory process *OM()* is introduced to hold the values, which includes a delay process *AT()* for each output.

```
process P_o =
  (? i1, ..., im; event OutEvent1, ..., OutEventn;
   ! o'1, ..., o'n;)
  (| (o1, ..., on) := P(i1, ..., im)
   | (o'1, ..., o'n) := OM
      (o1,...,on, OutEvent1, ..., OutEventn)
   |) where o1, ..., on; end;

process OM =
  (? ii1, ..., iin; event h1, ..., hn;
   ! oo1, ..., oon;)
  (| oo1 := AT(ii1, h1)
   | ...
   | oon := AT(iin, hn) |)

process AT =
  (? ii, event h; ! oo;)
  (| oo := ii cell h when h |)
```

2) Modeling propagation delays: Due to AADL specification, a process that is logically synchronous to a dispatch signal "*dispatch*" can be actually started later. Polychrony provides features for expressing activation (the clocks of signals). The main idea to model AADL unprecisely known time within a synchronous framework is to use additional inputs, called *activation conditions*, to model the propagation delays.

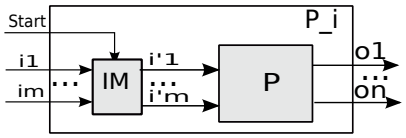

Fig. 3. Activation condition.

For each process P, an activation condition "*Start*" is introduced, and with SIGNAL process *IM()*, the input i is resynchronized with *Start* before entering synchronous program P (Figure 3).

```
pocess P_i =
  (? i1, ..., im; event Start; ! o1, ..., on;)
  (| (i'1, ..., i'n) := IM(i1, ..., im,Start)
   | (o1, ..., on) := P(i'1, ..., i'm)
   |) where i'1, ..., i'm; end;

process IM =
  (? i1, ..., im; event h; ! o1, ..., om;)
  (| o1 := AT(i1, h)
   | ...
   | om := AT(im, h) |)
```

3) Towards modeling time-based scheduling: Clocks of different threads or clocks in the same thread may be independent if timing relations are not precisely provided, which might cause problem of synchronization and non-determinism.

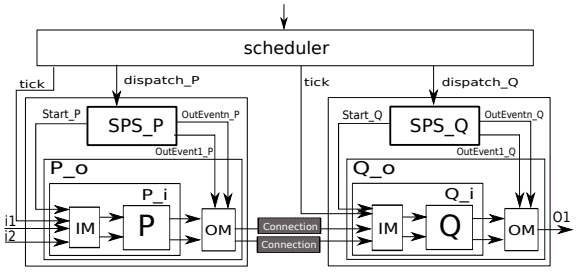

Fig. 4. Modeling asynchronous composition.

To solve this problem, we suppose that each thread P is associated with a timing environment SPS (Figure 4), which computes the timing control signals (i.e., *Start*, *OutEvent*, etc.), once it is activated (by *dispatch* event) by the scheduler. Figure 4 shows a composition of two threads P and Q presented in SIGNAL. Each component is activated by its corresponding activation event *Start*. Once SPS is triggered by the signal *dispatch*, it generates activation clock *Start* to activate the component, and computes the output available clock *OutEvent* that satisfies the specified clock properties.

Once these different timing semantics are bridged, now we can model the AADL threads and other components in

Polychrony. In the following, we will first present the modeling of a thread, then the communications between them.

B. Modeling AADL in Polychrony

1) Thread: Threads are the main executable and schedulable AADL components. To characterize its expressive capability, a thread $th \in \mathcal{TH}$, where $\mathcal{TH}$ is the set of thread components, encapsulates a functionality that may consist of ports $P =< p_1, p_2, \ldots, p_m >, p_i \in \mathcal{P}$, which declare the interface; connections set $C =< c_1, c_2, \ldots, c_n >$, where c_i is a connection $\mathcal{C}$ $c_i \in \mathcal{C}$; properties $R =< r_1, r_2, \ldots, r_k >$, $r_i \in \mathcal{R}$ is a property $\mathcal{R}$, which provide runtime aspects; behavior specification T/S that represents a transition system T over states S; and subcomponents Su that may be a set of subprograms or data.

$th = < P,\ C,\ R,\ Su,\ T/S >$

A thread is modeled within the following steps:

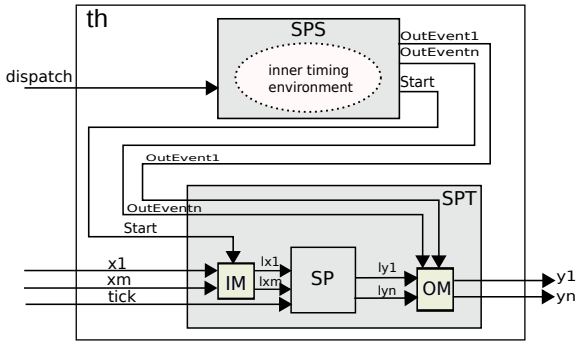

Fig. 5. Translation of AADL thread in SIGNAL

- An AADL thread th is firstly translated to a SIGNAL process SP, which contains the behaviors T/S (the detailed interpretation can be found in [17]) and the associated subcomponents Su and connections C. SP has the same input/output flows as th, plus an additional *tick* coming from the scheduler (Figure 5). Each input (resp. output) port $p_i \in P$ corresponds to an input (resp. output) signal. The notation $\mathcal{I}()$ is used to represent the translation.

  $SP = \mathcal{I}(P,\ C,\ Su,\ T/S)$
- According to the AADL specification, the input signals of a thread are transmitted from the sending threads at the output time of the ports. Due to the different timing semantics between AADL and SIGNAL, runtime aspects are translated by a SIGNAL process SPT (Figure 5), which plays the role of timing semantics interface between AADL and synchronous model. In SPT, two memory components *IM()* and *OM()* are added. The main functions of SPT with regard to SP include the memorization of signals and activation of thread.

  $SPT = (|IM|SP|OM|)$
- The execution of a thread is characterized by real-time features such as dispatch time and completion time. Due to this real-time control semantics, a new process SPS (Figure 5) is added, inside which the timing control signals are automatically computed once it is activated. SPS records all the temporal properties of the thread, and when it receives the scheduling signals (e.g., *dispatch*) from the real thread scheduler, it starts to calculate the timing signals (*Start, Completion, Deadline*...) for activating and completing the SPT process.

  $SPS = \mathcal{I}(R)$

2) Connection: Port connections are explicit relationships declared between ports or between port groups that enable the directional exchange of data and events among components. There are several categories of legal port connections: data port connections, event port connections, etc. In this paper, we will discuss the modeling of data port connections. Event and event data port connections support sampled connections and queues are required.

AADL provides three types of data port communication mechanisms between threads: *immediate*, *delayed* and *sampled* connections. For an immediate or delayed data port connection, both communicating periodic threads must be synchronized and dispatched simultaneously. The transmission time of a connection is specified by *Latency* property.

- ***Sampled***. A *Connection* process receives values from an output port when *OutEvent* is present. Since the communication takes a duration represented by *Latency*, an event *LatencyEvent* is introduced to specify the time at which the value has been sent to the destination. The received values cannot be used immediately: they are not available until input time, specified by *Input_Time* property and represented by an event *InEvent* in SIGNAL. Hence, another memory *AT()* process is added. Figure 6 gives a general interpretation of sampled data port connection.

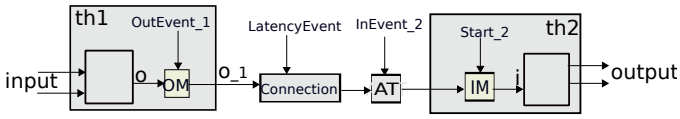

Fig. 6. AADL sampled data connection in SIGNAL processes

- ***Immediate***. Data transmission is initiated when the source thread completes and enters the suspended state. The output is transmitted to the receiving thread at the complete time of the sending thread ($OutEvent = Completion$) and available on the receiver side when $InEvent = Start$. The scheduler will dispatch the sender thread first, and ensure the receiver starts after the completion of the sender.
- ***Delayed***. The value from the sending thread is transmitted at its deadline ($OutEvent = Deadline$), and is available to the receiving thread at its next dispatch ($InEvent = dispatch$).

3) Processor and Scheduling: In AADL, a processor is an abstraction of hardware and software responsible for executing and scheduling threads. The property *Scheduling_Protocol* defines the way the processor will be shared between the threads of the application. The possible scheduling proto-

cols include: *Rate_Monotonic, Earliest_Deadline_First, Deadline_Monotonic, Least_Laxity_First*, *Highest_Priority_First* and any other user defined scheduling policy.

According to the AADL specification, a scheduler is needed for the simulation. It selects enabled tasks for execution according to the scheduling policy. A simple non-preemptive scheduler MS has been coded in SIGNAL manually for simulation. This scheduler takes into account events such as *dispatch, Start, Completion*, etc., for the scheduling of threads. More sophisticated schedulers, such as that provided by Cheddar [22] are expected to be integrated into the system.

4) Binding: A complete AADL system specification includes both software application and execution platform. The allocation of functionality onto architecture is specified, for example, the property *Actual_processor_binding* declares the processor binding property along with the AADL components and functionality to which it applies.

In the corresponding SIGNAL programs, the threads (SIGNAL processes) bound to the same processor are placed in a same process M, and they are controlled by the scheduler MS generated for the processor. The generated SIGNAL programs are annotated with allocation information. The function $Binding(th)$ provides the processor to which a thread th is bound. $\mathcal{I}(C)$ is a set of SIGNAL processes that are the interpretation of related connections. Each data port connection $c \in C$ could be modeled as described in the previous subsection. $\mathcal{I}(R)$ is a SIGNAL process that represents the interpretation of the related properties.

$$M = (\mathcal{I}(th_1) \mid \mathcal{I}(th_2) \mid \ldots \mid \mathcal{I}(th_m) \mid \mathcal{I}(R) \mid \mathcal{I}(C) \mid MS)$$
where $Binding(th_1) = Binding(th_2) = \cdots = Binding(th_m)$

All the SIGNAL processes bound to the same processor have the same SIGNAL pragma *RunOn i* [9], which enables the distribution of these processes onto the same processor *i*.

5) System: The system is the top-level component of the AADL model. A system can be organized into a hierarchy that can represent complex systems of systems as well as the integrated software and hardware of a dedicated application system.

A system $S =< P, C, R, Th >$ specifies the runtime architecture of an operational physical system, where $P =< p_1, p_2, \ldots, p_m >$ consists of the ports $p_i \in \mathcal{F}$ declaring the interface of the system, $C =< c_1, c_2, \ldots, c_n >, c_i \in \mathcal{C}$, represents the port connections among components, $R =< r_1, r_2, \ldots, r_k >, r_i \in \mathcal{R}$, specifies the properties among which additional allocation descriptive information are provided, and $Th =< th_1, th_2, \ldots, th_l >$ where $th_i \in \mathcal{TH}$, is a set of executable threads.

$\mathcal{I}(S) = (M_1 | M_2 | \ldots | M_{np} | \mathcal{I}(P) | SS)$
where $M_i = (\mathcal{I}(th_{i1}) | \mathcal{I}(th_{i2}) | \ldots | \mathcal{I}(R_i) | \mathcal{I}(C_i) | MS)$,
$Th_i =< th_{i1}, th_{i2}, \cdots >= \{th_{ij} \in \mathcal{TH}, Binding(th_{ij}) = processor_i\}$,
C_i, R_i are the sets of connections and properties related to Th_i,
np is the number of processors

The system S is transformed into a SIGNAL process, including a composition of container processes M_i, which includes the executable *thread* processes $\mathcal{I}(th_{ij})$ ($th_{ij} \in Th_i$), their connections $\mathcal{I}(C_i)$ and related properties $\mathcal{I}(R_i)$, as well as a global scheduler SS for activating each processor scheduler MS_i and ports interpretations $\mathcal{I}(P)$ (which are translated as input/output signals).

C. Simulation and verification

Once we have the complete model, based on SIGNAL, for AADL, we can carry out the synthesis so that a simulation model is generated. Profiling, verification and VCD-based (value change dump) simulation, which are briefly presented here, can be performed on this model.

In the framework of Polychrony, profiling, such as computing Worst-Case Execution Time (WCET), is used for purpose of performance evaluation [14]. The profiling process includes three steps: *temporal properties specification*, *temporal homomorphism*, and *co-simulation*. First, temporal properties, such as availability time and duration, are associated with SIGNAL variables, types, operators, etc. Second, we define a morphism as a series of transformations of SIGNAL models without changing their control part (the clock system of the model is preserved). Temporal properties are introduced in the morphism to reveal timing aspects of these models. The original SIGNAL models and their temporal homomorphisms are finally composed together for co-simulation. Figure 7 illustrates a schema of the co-simulation that has been carried out successfully. *PP* is the original SIGNAL program, whose inputs *I* are provided by *Inputs*. *T(PP)* is the temporal homomorphism of *PP* with regard to specified *Temporal properties* and a parameterization of *Library of cost functions*. *Date* provides date signals to *T(PP)* according to *I*. The input signals are synchronized to their corresponding date signals. Control values of *PP*, which decide specific traces of execution, are sent to *T(PP)* so that they have the same execution traces. Date signals of inputs and outputs of *T(PP)* are finally sent to *Observer* in order to obtain the simulation result *V*.

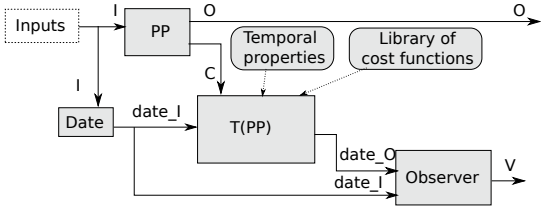

Fig. 7. The co-simulation of a SIGNAL program with regard to its temporal behavior.

Formal verification of functional aspects can be performed as a validation approach. AADL specifications are translated into SIGNAL, hence the model checking tool associated with Polychrony, called Sigali [19], can be used to check the system. Polynomial dynamical systems, as symbolic representation of the Sigali-based model checking, are obtained by compiling the SIGNAL programs. By formalizing the properties, with regard to *liveness, invariance, reachability*, etc.,

which are mainly based on computational tree logic (CTL), Sigali verifies if the properties hold for the overall system.

In addition to profiling and verification, a VCD-oriented simulation can also be performed. It aims at the visualization of value change during the program execution through graphical VCD viewers, such as GTK Wave. The visualization can be done dynamically at the same time as the execution.

IV. Related works

A number of AADL tools have been developed for system level design methodologies. As a free real-time scheduling tool, Cheddar [2] analyzes task temporal constraints of AADL systems. Ocarina [4] is an AADL-based code generation toolsuite.

In order to validate formal properties on AADL models, or for performance analysis, or verification of an AADL model, some other formal models could be used to represent the AADL models. It has been widely studied in AADL2Fiacre [8], SLIM [20], etc. In these projects, AADL is modeled using an intermediate language for verification of high level models. We follow a similar approach: SIGNAL is adopted as a common formalism, which can be transformed into polynomial equations associated with a model checking tool. An advantage of our approach is that formal verification, simulation and analysis can be directly carried out on this common formalism, without supplementary translation into some other formalism.

Due to the limitation of pure asynchronous design pattern, synchronous modeling of asynchronous systems has been an important issue. In [11], a generic semantic model for synchronous and asynchronous computation is defined, after that, the attention is focused on implementing communication mechanisms. [18] relies on the MARTE [3] Time Model and the operational semantics of its companion language CCSL [6], to equip UML activities with the execution semantics of an AADL specification. It makes efforts to build a generic simulator specifically for AADL, but targeting a formal analyzable language remains a perspective. Although AADL2SYNC [1] also models AADL using a synchronous language, it considers a purely synchronous model of computation (that of Lustre), in which clocks need to be totally ordered. Its expressive capability is limited compared to multi-clocked MoC considered in SIGNAL.

V. Conclusion

In this paper, we present a model of a part of AADL software and hardware concepts, such as thread, connection, and processors, based on the SIGNAL's polychronous model of computation. Time-related concepts in AADL, such as synchronization, delay, period, etc., are abstracted by using discrete logical clocks, which can be independent or dependent. Thus, AADL time domain is mapped onto SIGNAL clocks. Then, independent or dependent clocks are partially correlated by schedulers and allocation specification for simulation. An overview of the model description and simulation results of an avionic case study, SDSCS system, can be found in [23] to demonstrate our approach. [23] focuses on the issue of composing, integrating and simulation heterogeneous models (AADL model and Simulink model) in a system co-design flow, and in this paper, we mainly present the modeling for AADL in Polychrony.

This modeling supports the virtual prototyping and formal validation of component-based embedded architectures, and enables software synthesis for the purpose of early phase simulation. A refinement of this modeling in consideration of more AADL concepts, including more temporal properties, memory, data access, etc., is in progress. We are also interested in introducing controller synthesis for the partition-level scheduling [19] and distributed code generation using Syndex [5].

References

[1] AADL2SYNC, Verimag. http://www-verimag.imag.fr/~synchron/index.php?page=aadl2sync.

[2] Cheddar Project. http://beru.univ-brest.fr/~singhoff/cheddar.

[3] MARTE. http://www.omgmarte.org/.

[4] Ocarina Project. http://ocarina.enst.fr/.

[5] Syndex. http://www-rocq.inria.fr/syndex/.

[6] C. André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Research Report RR-6925, INRIA, 2009.

[7] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone. The synchronous languages twelve years later. In *Proceedings of the IEEE*, pages 64–83, 2003.

[8] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang, and F. Vernadat. Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In *ERTS*, 2008.

[9] L. Besnard, T. Gautier, P. Le Guernic, and J.-P. Talpin. Compilation of Polychronous Data Flow Equations. In S. Shukla and J.-P. Talpin, editors, *Synthesis of Embedded Software*, pages 1–40. Springer, 2010.

[10] F. Boussinot and R. de Simone. The ESTEREL Language. *Proceedings of the IEEE*, 79, 1991.

[11] N. Halbwachs and S. Baghdadi. Synchronous Modelling of Asynchronous Systems. In *EMSOFT'02*, 2002.

[12] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language LUSTRE. In *Proceedings of the IEEE*, pages 1305–1320, 1991.

[13] INRIA ESPRESSO team. Polychrony. http://www.irisa.fr/espresso/Polychrony/.

[14] A. Kountouris and P. Le Guernic. Profiling of SIGNAL Programs and Its Application in the Timing Evaluation of Design Implementations. In *Proceedings of the IEE Colloq. on HW-SW Cosynthesis for Reconfigurable Systems*, 1996.

[15] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming Real-Time Applications with SIGNAL. *Proc. of the IEEE*, 79, 1991.

[16] P. le Guernic, J.-P. Talpin, and J.-C. le Lann. Polychrony for System Design. *Journal for Circuits, Systems and Computers*, 12, 2002.

[17] Y. Ma, J.-P. Talpin, and T. Gautier. Interpretation of AADL Behavior Annex into synchronous formalism using SSA. In *ESA2010*, pages 2361–2366. IEEE Computer Society, 2010.

[18] F. Mallet, C. André, and J. De Antoni. Executing AADL Models with UML/MARTE. In *ICECCS '09*, pages 371–376, Washington, DC, USA, 2009. IEEE Computer Society.

[19] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. Synthesis of Discrete-Event Controllers based on the SIGNAL Environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4), Oct. 2000.

[20] V. Nguyen, T. Noll, and M. Odenbrett. Slicing AADL Specifications for Model Checking. In *NFM'10*, pages 217–221. NASA, April 2010.

[21] SAE Aerospace. Architecture Analysis and Design Language (AADL). *SAE AS5506A*, 2009.

[22] F. Singhoff and A. Plantec. AADL Modeling and Analysis of Hierarchical Schedulers. *ACM SIGAda Letters*, 27(3):41–50, 2007.

[23] H. Yu, Y. Ma, Y. Glouche, J.-P. Talpin, L. Besnard, T. Gautier, P. Le Guernic, A. Toom, and O. Laurent. System-level Co-simulation of Integrated Avionics Using Polychrony. In *SAC'11*, Taiwan, March 2011.